IBM Watson IoT

IoT for insurance:

A case for

Complex event processing

Markus Müller

email:markus_mueller@de.ibm.com

1/18/17

Motivation

Top 5 Benefits of IoT to the Financial Services Industry

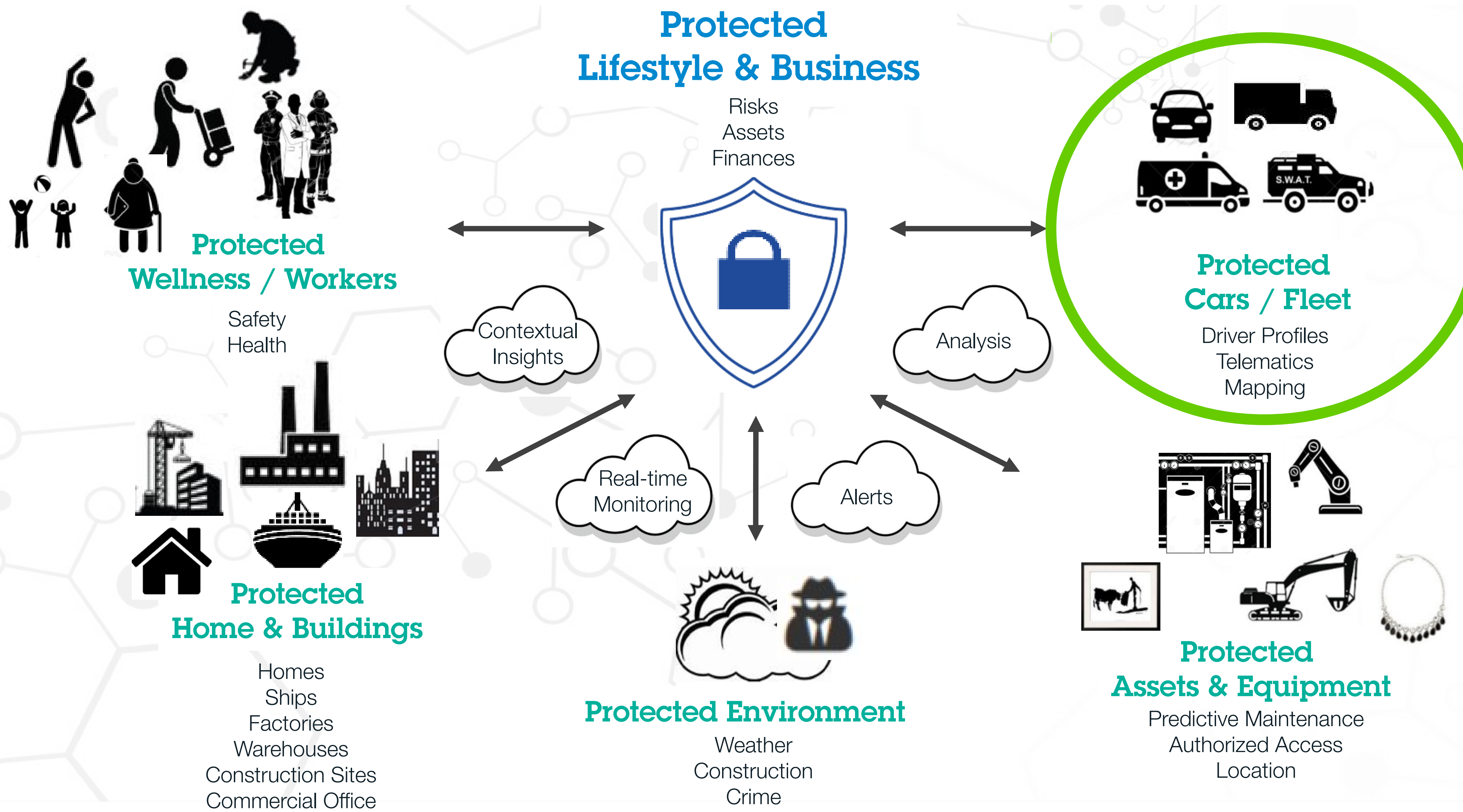
IBM Watson IoT

What we learn from the physical world will transform several industries, including the Insurance Sector in which IoT will have one of the greatest impacts.

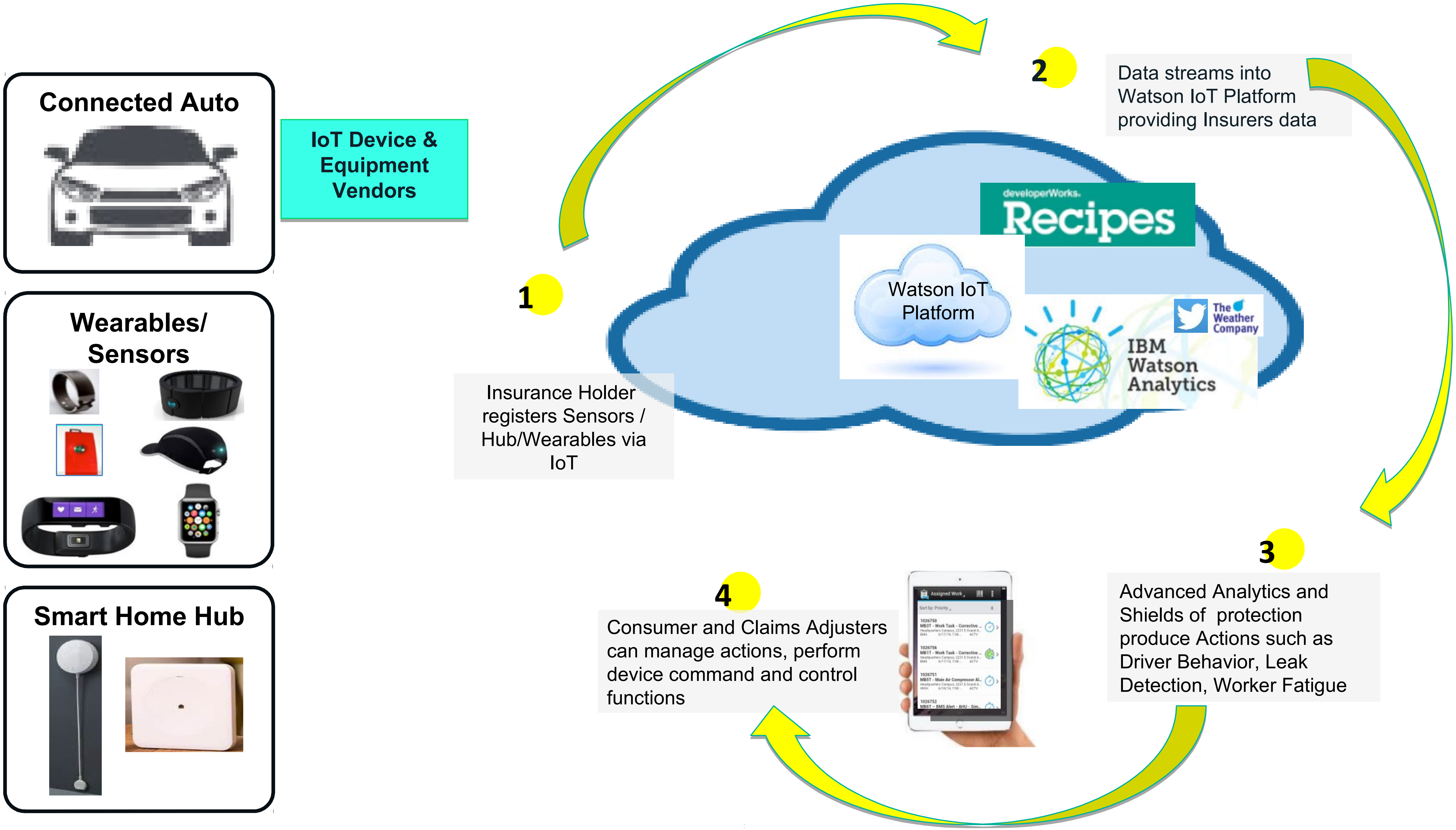
[1] *First Notice of Loss (FNOL)*

Insurance Industry Transformation to Proactive Protection & Claims Avoidance

Carriers are transforming their historic risk assessment models by proactively mitigating risk through real time alerts and analytics



IBM IoT for Insurance – How it works



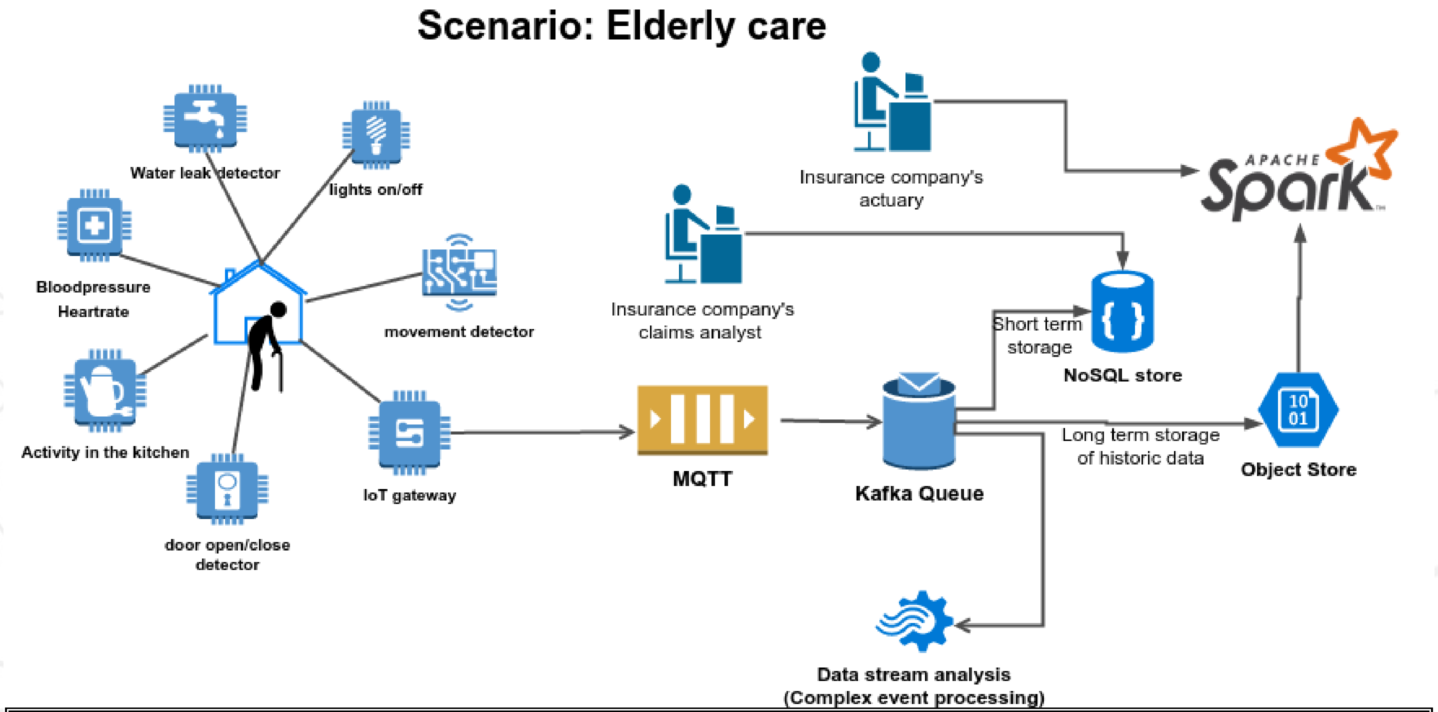
Insurance shield example

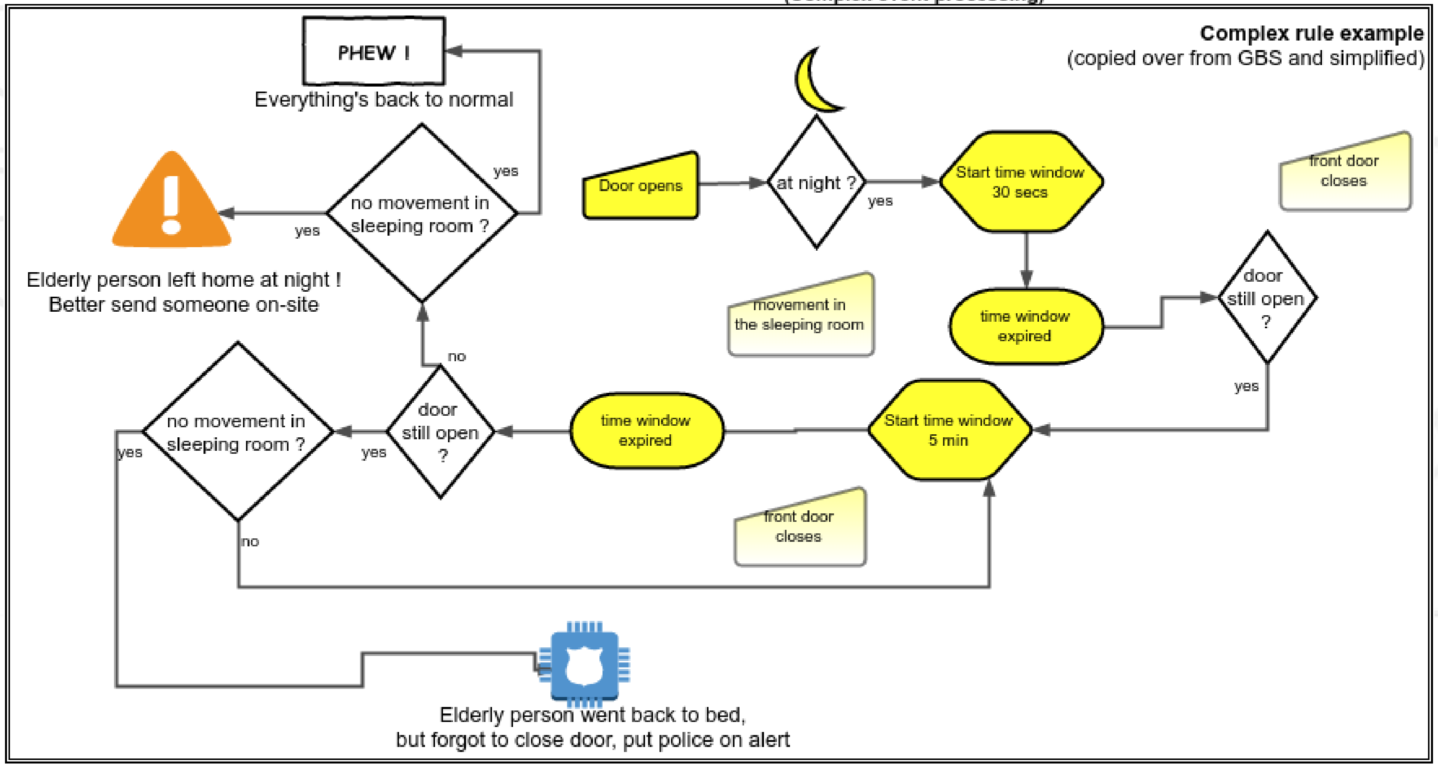
"Door opens at night"

Evaluating this rather convoluted shield gives rise to Complex Event Processing:

Requirement:
- Stateful processing
- Sliding windows

Other shields like "heart rate monitoring" make use of tumbling windows.

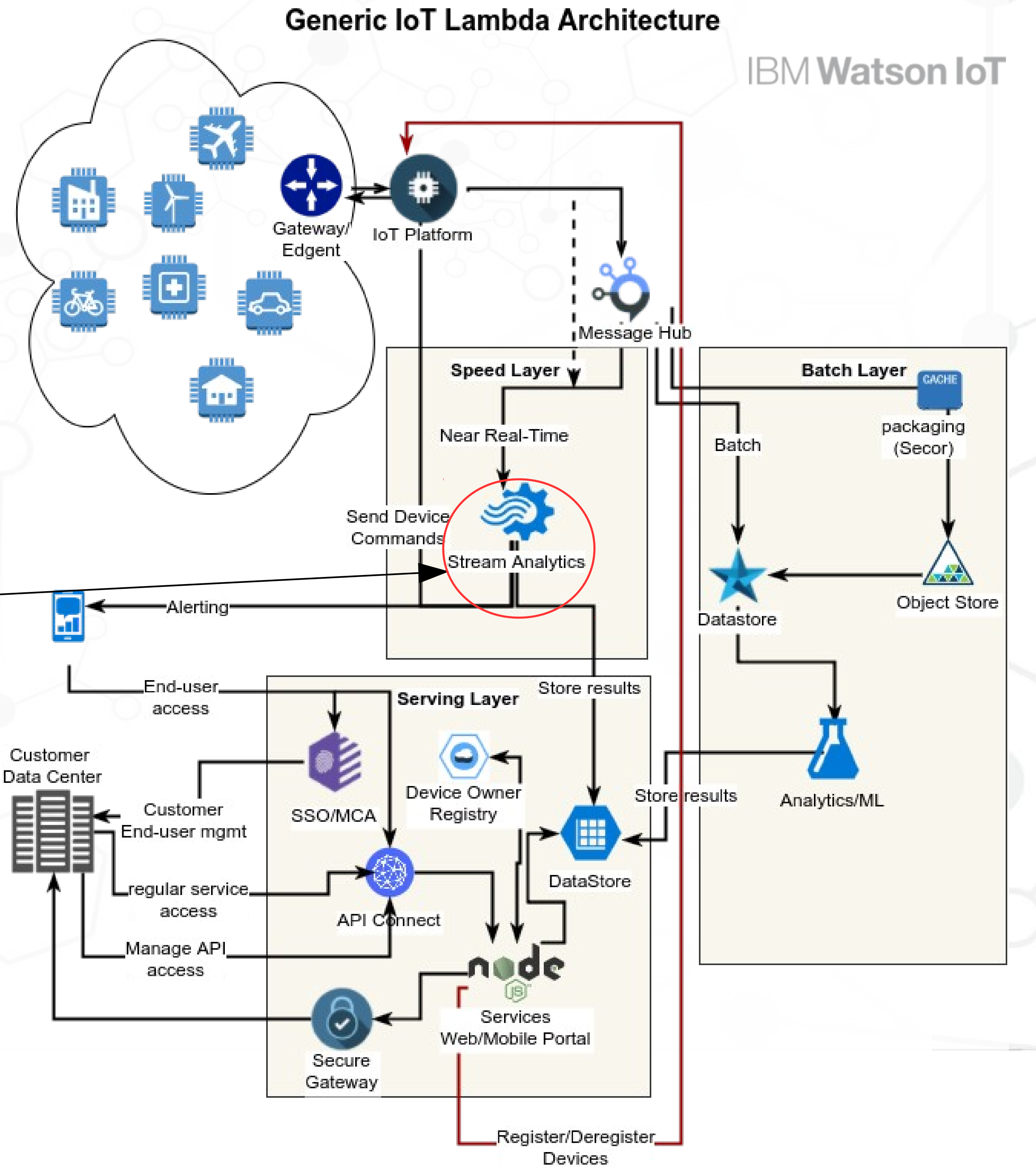




See also here
https://i.stack.imgur.com/mm06A.jpg
For windowing in data stream processing

IoT Reference Architecture:

Components and Data Flow



Complex Event Processing typically needs to be executed near real-time, so in a Lambda architecture it takes place **here**

Technical background

Complex event processing in a nutshell

IBM Watson IoT

Events

exhibit the following characteristics. They

- Are **Immutable**
- Have **strong temporal constraints**
- Have a **managed lifecycle** (become "stale" after some time) and thus as subject to a **sliding window** of 'validity'.

In the IOT case events have some 'sparseness' traits, so

- There is a **high volume** of events.
- **Patterns** are more important than invidual events.

Event Processing

is a method of tracking and analyzing (processing) streams of information (data) about things that happen (events),[1] and deriving a conclusion from them.

Complex Event Processing

or CEP, is event processing that combines data from multiple sources to infer events or patterns that suggest more complicated circumstances. The goal of complex event processing is to identify meaningful events (such as opportunities or threats) and respond to them as quickly as possible.

Traits/functional requirements

- Allow detection, correlation, aggregation and composition of events.
- Support processing of Streams of events.
- Support temporal constraints in order to model the temporal relationships between events.
- Support sliding windows of interesting events.
- Support a properly scoped unified clock - in the case mentioned in the deck the proper scope would be all sensors in the elderly person's apartment.
- Support the required volumes of events for CEP use cases.
- Support MQTT for event input into the engine.

For background see wikipedia.org

Non-functional requirements for complex event processing

IBM Watson IoT

“At least once” guarantee

No IoT event is lost before being processed. IoT events must be retained/buffered until handled otherwise we could lose events, resp. do not sent out actions
Note that this implies **fault tolerance**, i.e. restartability while preserving its internal state.

Partial ordering guarantee

Events for a particular group of devices (for example grouped by ownership) are delivered and processed in order.
→ See elderly care example.

“Exactly once” guarantee

No action must be triggered twice.

Scalability

CEP must scale with number of users and event rate.

Latency

Actions should be triggered in < 60 secs.

Supports modular programming model

Adding/Modifying a shield (specific complex event evaluation) should not affect other shields (Modular development, test and deployment)

Reduce Total cost of ownership

Favor services available on Bluemix over COTS over special components over customer code in terms of development/maintenance/operations costs. **Maturity**/Reliability of components is subsumed here

Ease of operations

Any component must be equipped with proper monitoring for 7x24 Ops. Measures also the effort to create runbooks and monitoring.

Automated instance deployment

This either means a multi-tenant solution where instance creation doesn't affect deployment or we need a fully automated true deployment. The former requires tenant isolation, the latter either a bluemix service or well designed setup 'scripts'. Upgrade without downtime is subsumed here.

Maturity

Prefer standard, well established architectural patterns and components over FOAK to reduce risks.

Note: Left side requirements are MUST, right side count as SHOULD

For background and terminology
see https://dzone.com/articles/kafka-clients-at-most-once-at-least-once-exactly-o

IoT for insurance use case volumetrics – one year estimate

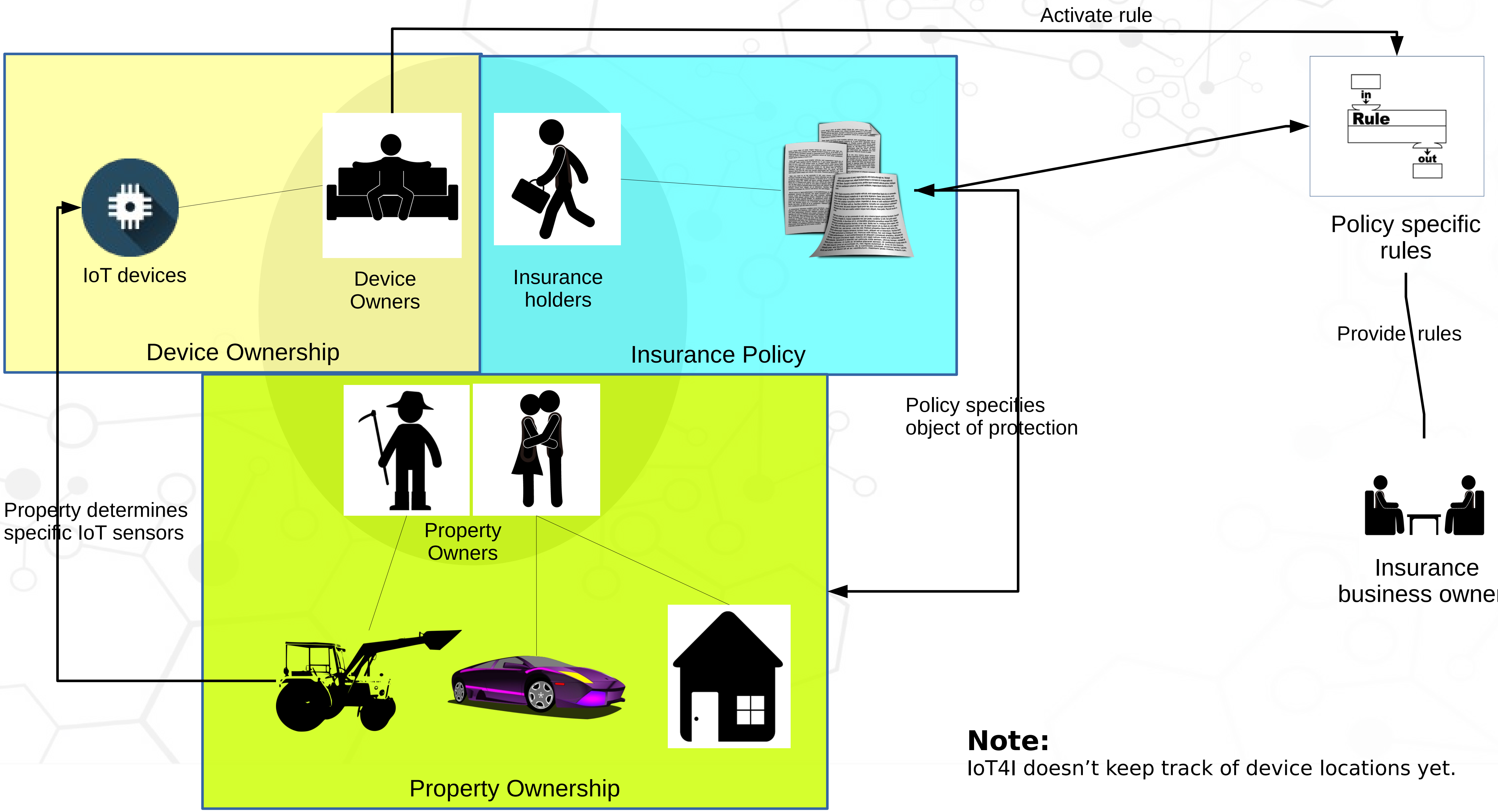
IBM Watson IoT

… blanked out confidential data …

AVG messages per sec	115,000.00	46,000.00	11,500.00	
AVG data per sec (MB)	8.87	3.55	0.89	

Persistent data in IoT4I – relationships in a single deployment

IBM Watson IoT



Note:
IoT4I doesn't keep track of device locations yet.

Persistent data in IoT4I – pattern matching state engine data

IBM Watson IoT

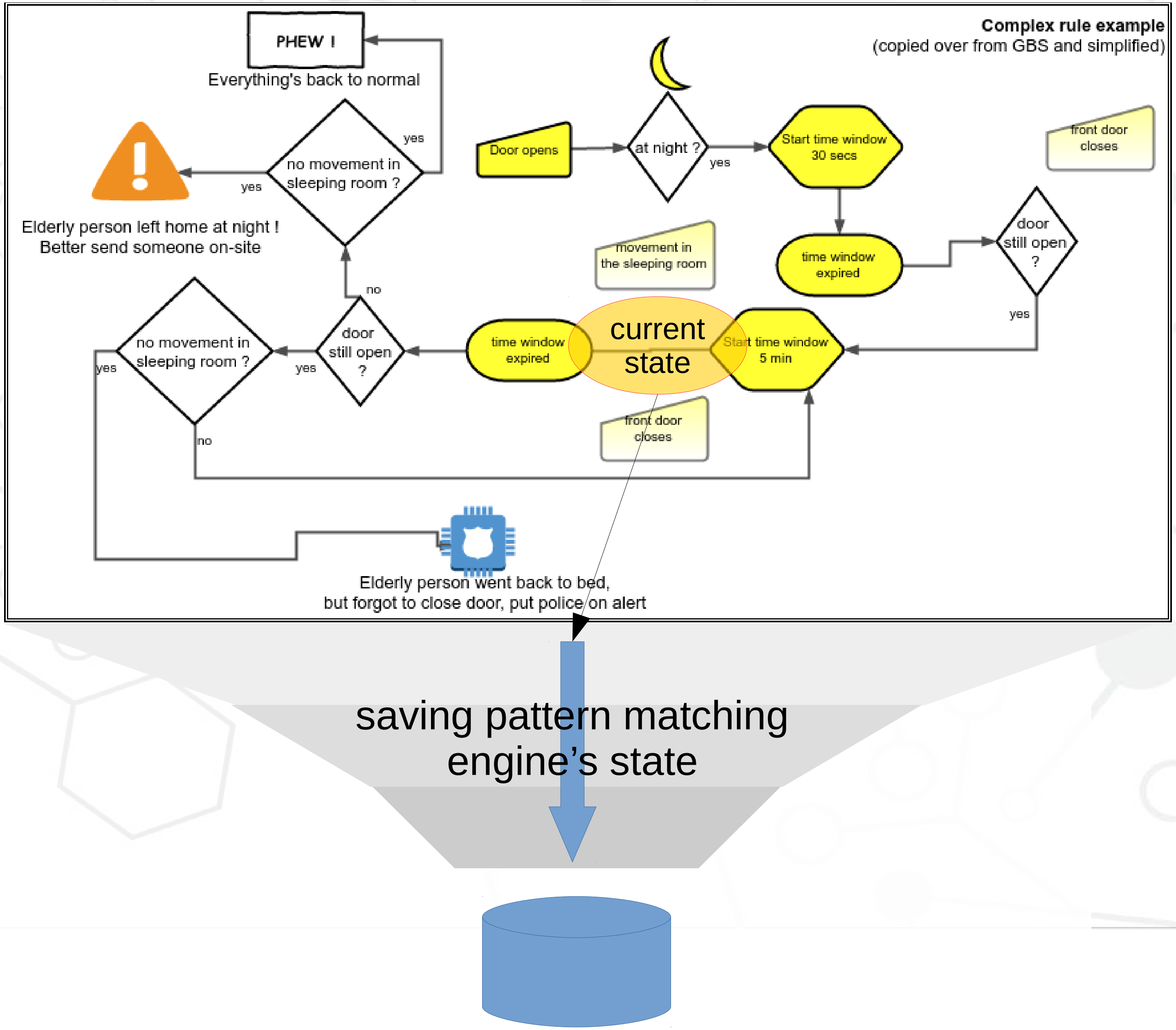
Considerations:

Saving state vs. periodic checkpointing vs. saving event offsets

- Saving state requires a write operation after applying operators in the pattern matching, so we get many small write operations.
- Saving event offset of the event when the state engine was in initial state requires the event source to maintain persistency and runs the risk of duplicate actions
- Periodic Checkpointing covers the middle ground

What datamodel is the best fit ?

- Saving offsets requires minimal amount of data, basically a triple of shield, user and queue offset to the last terminal event that causes a transition of the state engine to initial state.
- Saving state or checkpointing results in unstructured data best suited for a NoSQL database.



Selecting candidates

Open Source frameworks for handling stream data

	Flume	NiFi	Gearpump	Apex	Kafka Streams	Spark Streaming	Storm	Storm + Trident	Samza	Flink	Ignite Streaming	Beam [*GC DataFlow]
Current version	1.6.0	0.6.1	incubating	3.3.0	0.9.0.1* [available in 0.10]	1.6.1	1.0.0	1.0.0	0.10.0	1.0.2	1.5.0	incubating
Category	DC/SEP	DC/SEP	SEP	DC/ESP	ESP	ESP	ESP/CEP	ESP/CEP	ESP	ESP/CEP	ESP/CEP	SDK
Event size	single	single	single	single	single	micro-batch	single	mini-batch	single	single	single	single
Available since [incubator since]	June 2012 [June 2011]	July 2015 [Nov 2014]	[Mar 2016]	Apr 2016 [Aug 2015]	Apr 2016 [July 2011]	Feb 2014 [2013]	Sep 2014 [Sep 2013]	Sep 2014 [Sep 2013]	Jan 2014 [July 2013]	Dec 2014 [Mar 2014]	Sep 2015 [Oct 2014]	[Feb 2016]
Contributors	26	67	19	53	160	838	207	207	48	159	56	80
Main backers	Apple Cloudera	Hortonworks	Intel Lightbend	Data Torrent	Confluent	AMPLab Databricks	Backtype Twitter	Backtype Twitter	LinkedIn	dataArtisans	GridGain	Google
Delivery guarantees	at least once	at least once	exactly once at least once [with non-fault-tolerant sources]	exactly once	at least once	exactly once at least once [with non-fault-tolerant sources]	at least once	exactly once	at least once	exactly once	at least once	exactly once*
State management	transactional updates	local and distributed snapshots	checkpoints	checkpoints	local and distributed snapshots	checkpoints	record acknowledgements	record acknowledgements	local snapshots distributed snapshots [fault-tolerant]	distributed snapshots	checkpoints	transactional updates*
Fault tolerance	yes [with file channel only]	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes*
Out-of-order processing	no	no	yes	no	yes	no	yes	yes	yes [but not within a single partition]	yes	yes	yes*
Event prioritization	no	yes	programmable	programmable	programmable	programmable	programmable	programmable	yes	programmable	programmable	programmable
Windowing	no	no	time-based	time-based	time-based	time-based	time-based count-based	time-based count-based	time-based	time-based count-based	time-based count-based	time-based
Back-pressure	no	yes	yes	yes	N/A	yes	yes	yes	yes	yes	yes	yes*
Primary abstraction	Event	FlowFile	Message	Tuple	KafkaStream	DStream	Tuple	TridentTuple	Message	DataStream	IgniteDataStreamer	PCollection
Data flow	agent	flow [process group]	streaming application	streaming application	process topology	application	topology	topology	job	streaming dataflow	job	pipeline
Latency	low	configurable	very low	very low	very low	medium	very low	medium	low	low [configurable]	very low	low*
Resource management	native	native	YARN	YARN	Any process manager [e.g. YARN, Mesos, Chef, Puppet, Salt, Kubernetes, ...]	YARN Mesos	YARN Mesos	YARN Mesos	YARN	YARN	YARN Mesos	integrated*
Auto-scaling	no	no	no	yes	yes	yes	no	no	no	no	no	yes*
In-flight modifications	no	yes	yes	yes	yes	no	yes [for resources]	yes [for resources]	no	no	no	no
API	declarative	compositional	declarative	declarative	declarative	declarative	compositional	compositional	compositional	declarative	declarative	declarative
Primarily written in	Java	Java	Scala	Java	Java	Scala	Clojure	Java	Scala	Java	Java	Java
API languages	text files Java	REST [GUI]	Scala Java	Java	Java	Scala Java Python	Scala Java Clojure Python Ruby	Java Python Scala	Java	Java Scala Python	Java .NET C++	Java*
Notable users	Meebo Sharethrough SimpleGeo	N/A	Intel Levi's Honeywell	Capital One GE Predix PubMatic	N/A	Kelkoo Localytics AsiaInfo Opentable Faimdata Guavus	Yahoo! Spotify Groupon Flipboard The Weather Channel Alibaba Baidu Yelp	Klout GumGum CrowdFlower	LinkedIn Netflix Intuit Uber	King Otto Group	GridGain	N/A

Taken from Ian Hellström, https://databaseline.wordpress.com/2016/03/12/an-overview-of-apache-streaming-technologies/

Stream Analytics frameworks – narrowing down the list of candidates

An overview article* in Predictive Analytics Today mentions five leading frameworks for Stream Analytics

- Apache Flink
- Apache Spark Streaming
- Apache Samza
- Apache Storm
- IBM Streams (taken as roughly equivalent to Apache Flink, so most of the investigation carries over)

From that list we derived the following approaches

1. Kafka shield engine – similar to Samza and the more recent Kafka Streams but based on Javascript to allow for the reuse of existing shield code and relying on Bluemix for resource management
2. OpenWhisk shield dispatcher on Kafka – similar to the “Kafka shield engine” but with OpenWhisk for resource management
3. Apache Flink with MQTT connector
4. Apache Spark Streaming on Kafka

And we added the last approach

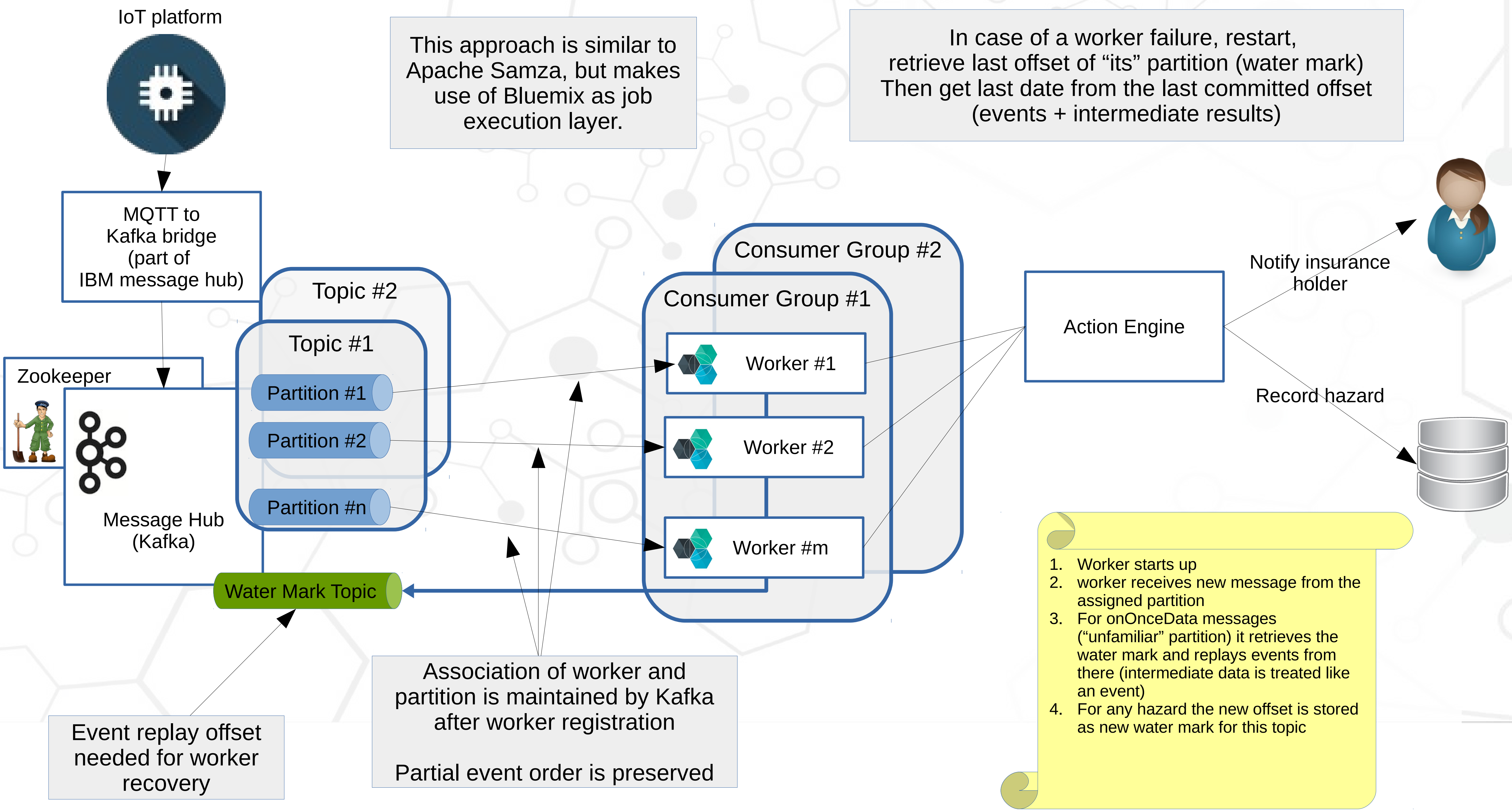
5. Agent framework based shield dispatcher on MQTT

was included to open the door for collaboration with our sister project ‘IoT for automotive’

While CEP libraries like WSO2’s Siddhi can work with Storm, it can also be based on Kafka directly, so Storm is only adding complexity; hence I stopped further investigating Storm as viable technology.

*See Imanuel Muller in http://www.predictiveanalyticstoday.com/top-open-source-commercial-stream-analytics-platforms/

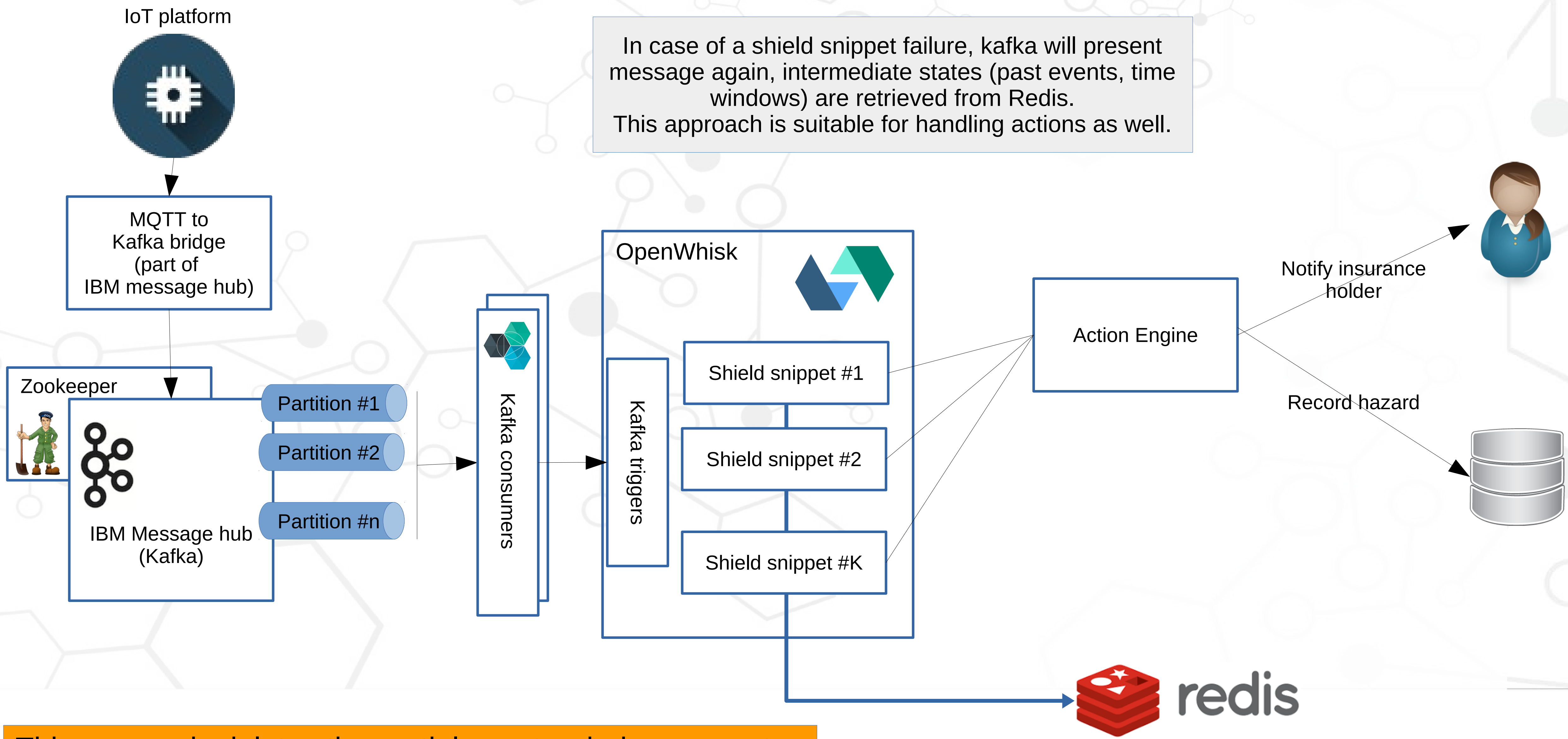
Approach #1: Kafka shield engine



Approach #2: OpenWhisk shield dispatcher on Kafka

IBM **Watson IoT**

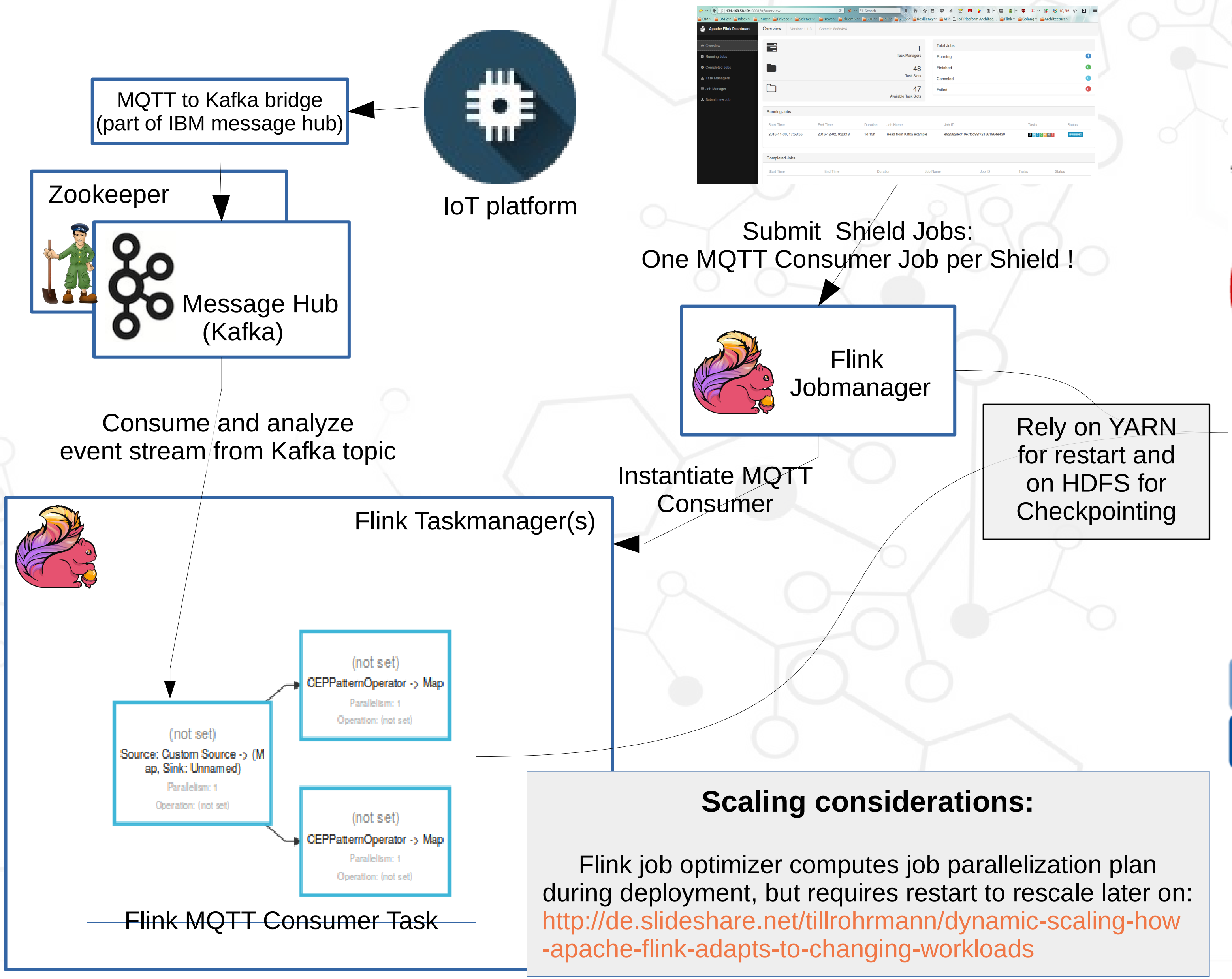
In case of a shield snippet failure, kafka will present message again, intermediate states (past events, time windows) are retrieved from Redis.
This approach is suitable for handling actions as well.

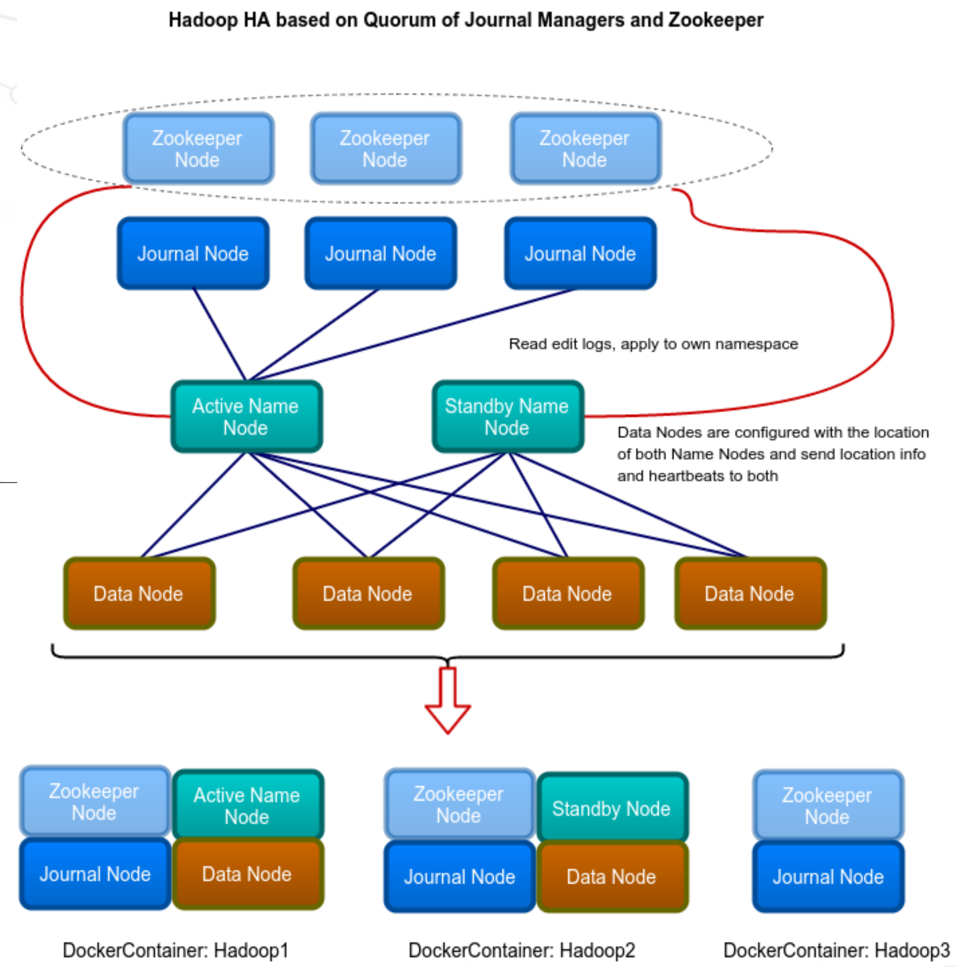


This approach violates the partial event ordering guarantee

Approach #3: Apache Flink with Kafka connector

IBM Watson IoT





Scaling considerations:

Flink job optimizer computes job parallelization plan during deployment, but requires restart to rescale later on: http://de.slideshare.net/tillrohrmann/dynamic-scaling-how-apache-flink-adapts-to-changing-workloads

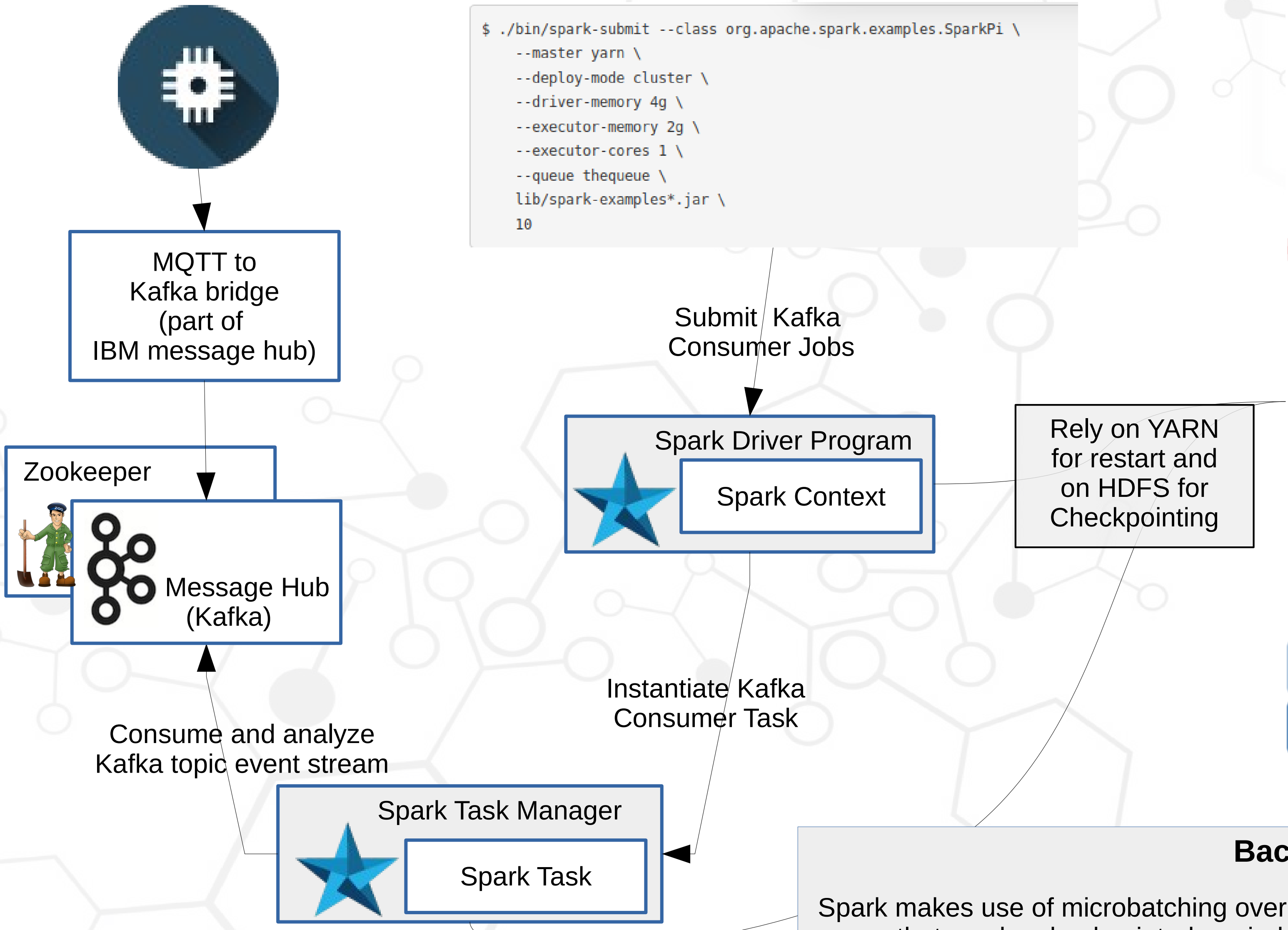
MQTT Backpressure considerations:

Flink checkpoints every 5 secs by default, IoT4I expects < 120 evts/sec, with a dedicated IoT platform MQTT broker we're well below the limit of 5000 retained messages (Watson IoTP)

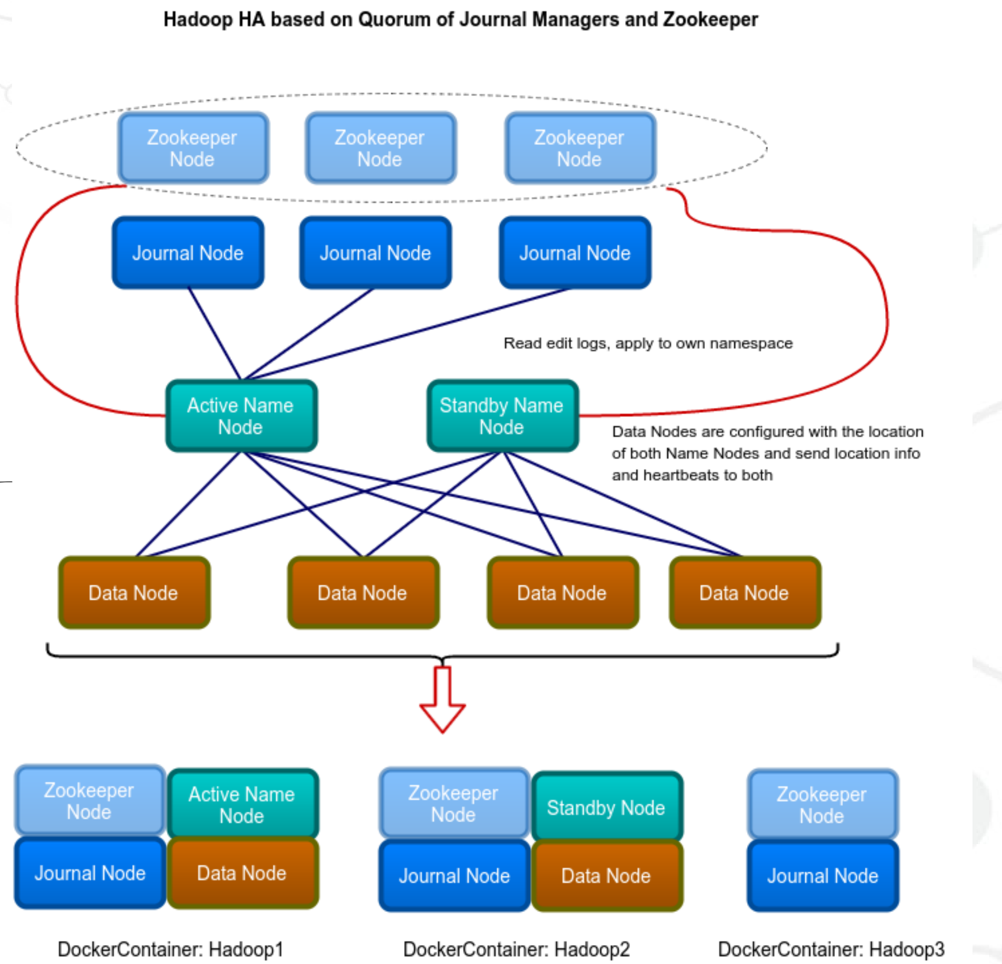
See the following NewRelic presentation for a real life data aggregation show case
http://berlin.flink-forward.org/wp-content/uploads/2016/07/Ron-Crocker-Evaluating-Streaming-Framework-Performance-for-a-Large-Scale-Aggregation-Pipeline.pdf

Approach #4: Apache Spark Streaming with Kafka connector

IBM Watson IoT




```
$ ./bin/spark-submit --class org.apache.spark.examples.SparkPi \
    --master yarn \
    --deploy-mode cluster \
    --driver-memory 4g \
    --executor-memory 2g \
    --executor-cores 1 \
    --queue thequeue \
    lib/spark-examples*.jar \
    10
```



Backpressure considerations:

Spark makes use of microbatching over D(istributed)Streams, an ensemble of Resilient Distributed Datasets that can be checkpointed; period depends on the DStream's granularity (default interval 10sec; checkpointing is a multiple of that, so >= 10 secs). IoT4I expects < 120 evts/sec, with a dedicated IoT platform MQTT broker + Kafka Hub we're well below the event rate that can be consumed by a regular Spark Streaming Task.
BTW, the current MQTT Spark consumer (Bahir) does not support back pressure, so Kafka is required here

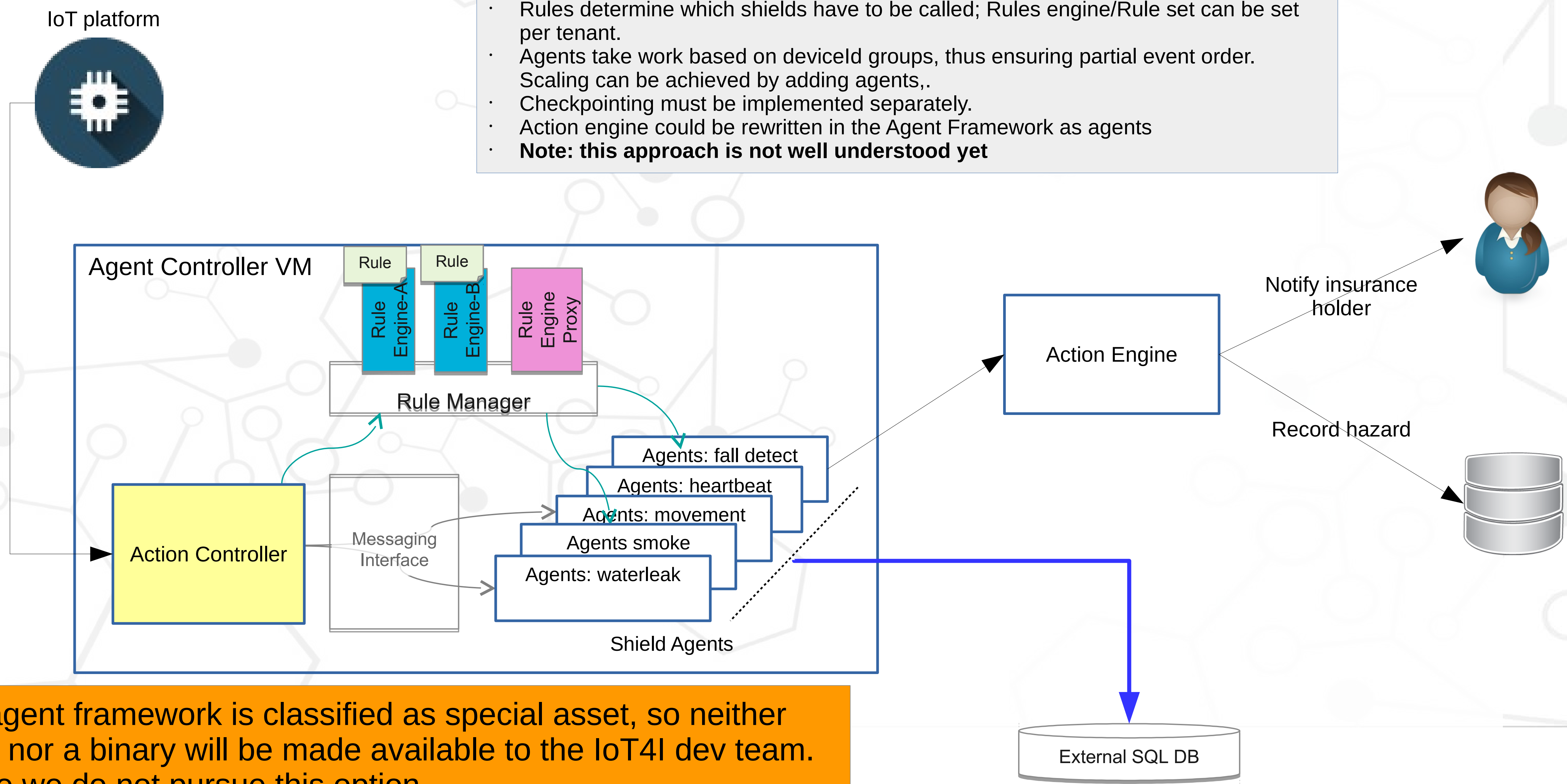
For background see
http://spark.apache.org/docs/latest/streaming-programming-guide.htm
http://spark.apache.org/docs/latest/running-on-yarn.html

Approach #5: Agent framework based shield dispatcher on MQTT

IBM Watson IoT

Considerations

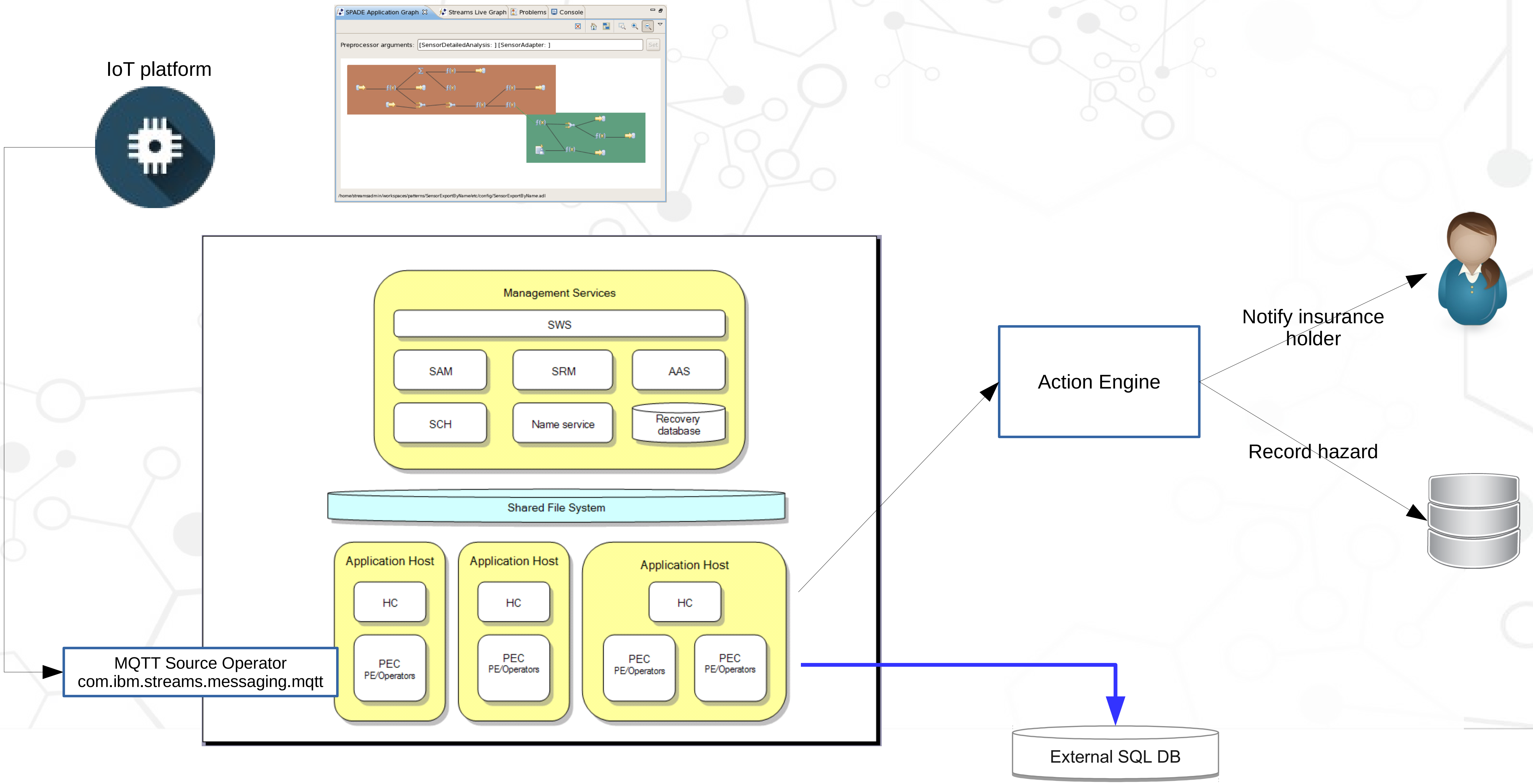
- Rules determine which shields have to be called; Rules engine/Rule set can be set per tenant.
- Agents take work based on deviceId groups, thus ensuring partial event order. Scaling can be achieved by adding agents,.
- Checkpointing must be implemented separately.
- Action engine could be rewritten in the Agent Framework as agents
- **Note: this approach is not well understood yet**



The agent framework is classified as special asset, so neither code, nor a binary will be made available to the IoT4I dev team. Hence we do not pursue this option

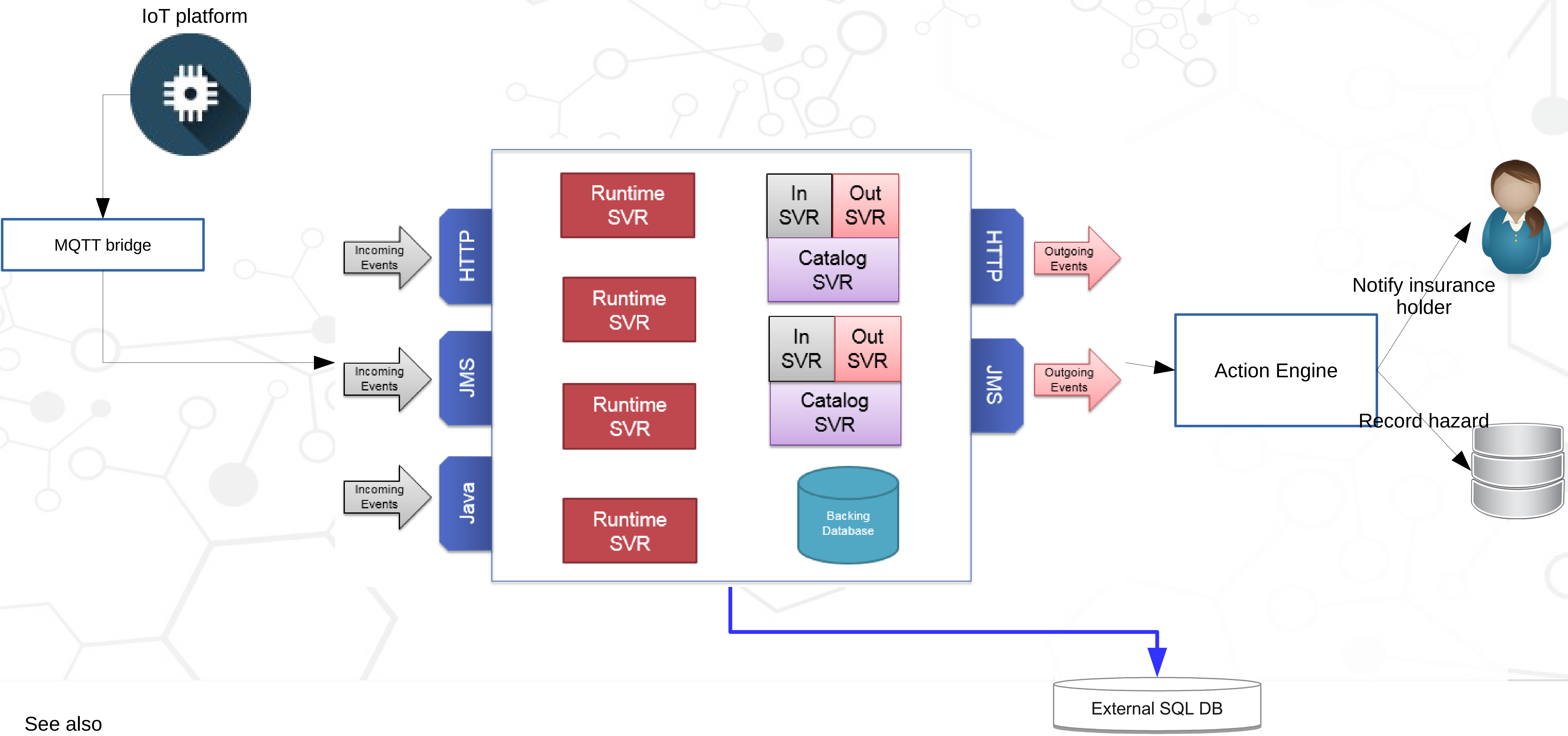
Approach #6: IBM Streams

IBM Watson IoT



Approach #7: ODM Decision Server Insights

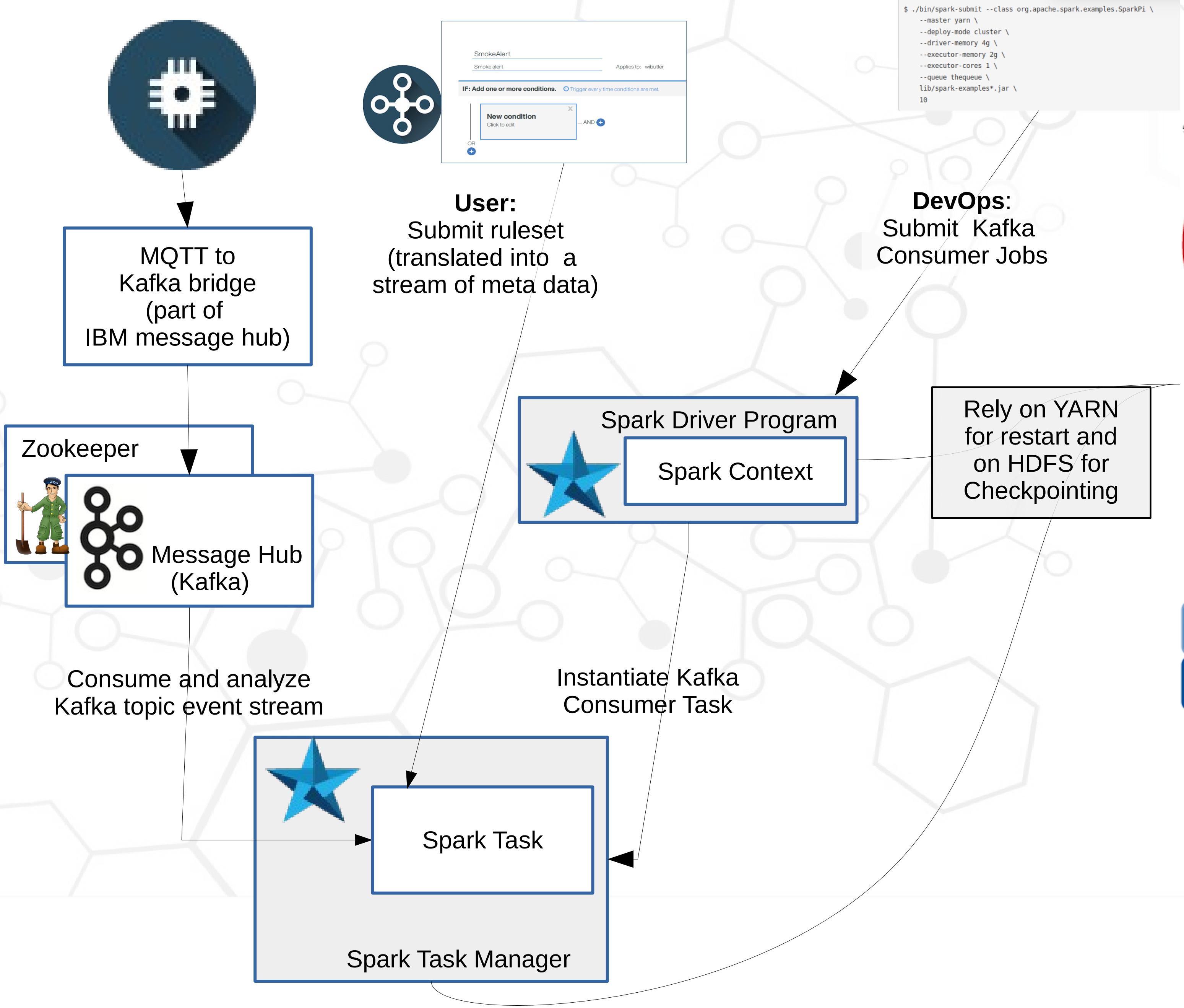
IBM Watson IoT

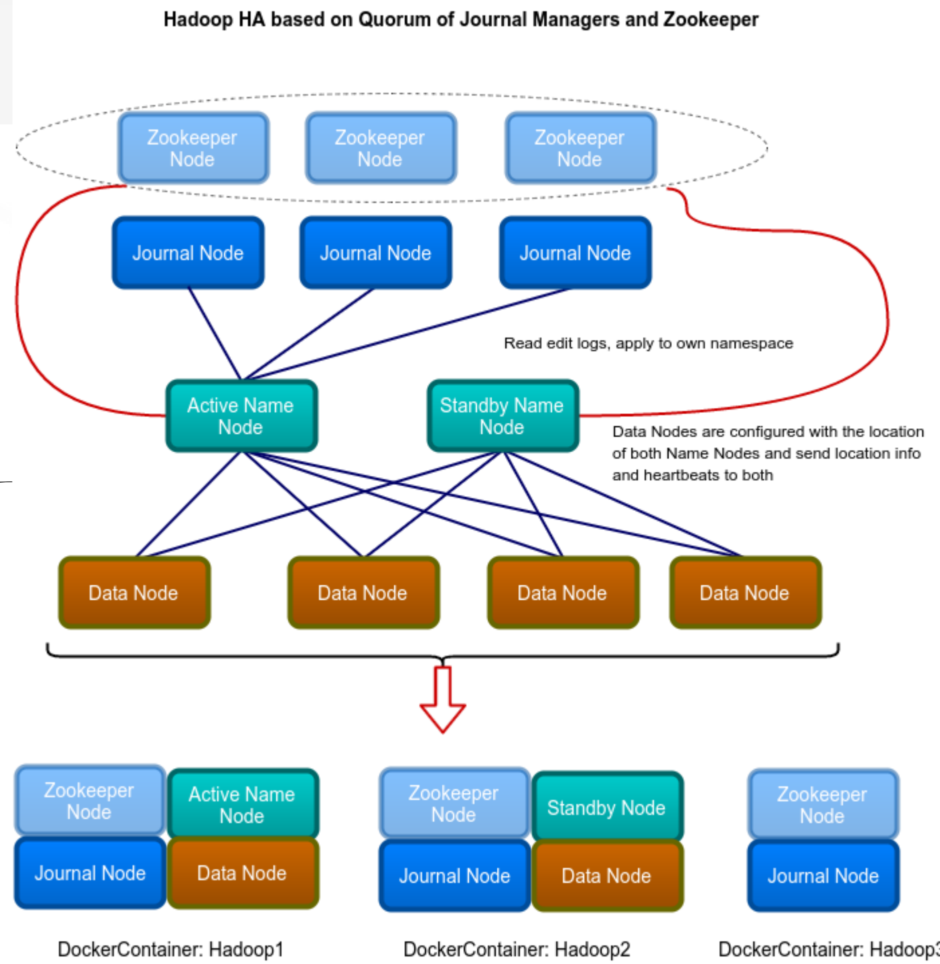


See also
https://hub.jazz.net/project/dselman/InternetOfInsight/overview

Approach #8: Real Time Insights

IBM Watson IoT





RTI is built on Spark Streaming

Payload is processed according to Metadata. This is possible due to

For background see
http://spark.apache.org/docs/latest/streaming-programming-guide.htm
http://spark.apache.org/docs/latest/running-on-yarn.html

Decision Matrix "Must Have requirements"

IBM Watson IoT

	"At least once" Fault tolerance, Back pressure	Partial ordering guarantee	"Exactly once"	Scalability (without affecting runtime)	Latency	Functional elements Stateful operators + Windows
Kafka shield engine	Yes	Yes	Maybe	By adding partitions and workers (online)	Low	No stateful sliding or tumbling windows
OpenWhisk shield dispatcher on Kafka	Yes	No	No	By adding partitions (online)	Medium due to OpenWhisk	No stateful sliding or tumbling windows
Apache Flink with Kafka connector	Yes	Yesb	Yes	Redeploy Flink CEP Program with higher parallelization (red-black)	Low	Yes
Spark Streaming on Kafka	Yes	Yes	No	Redeploy Spark Streaming program (red-black)	Medium due to micro-batching (for us it's okay)	Limited There are unmaintained packages available
Agent framework shield dispatcher on MQTT	Yes	Yes	No	Yes	*Low (according to their dev team)*	No stateful sliding or tumbling windows, no operators
IBM streams	Yes	Yes	Yes	Yes (online ?)	Low	Yes
ODM Decision Server Insights			Not known	Yes	Not known	Yes
Real Time Insight			No	Yes	See Spark Streaming	No stateful sliding or tumbling windows

Work in progress

Decision Matrix "Should Have requirements"

IBM Watson IoT

	Modular CEP programming/ definition model	Modification without outage	Rule editor (no programming required)	Dev/Test Effort	Low TCO (Ops + Maintenance)	Automated Deployment	Maturity	Cost
Kafka shield engine	Yes, shield expressed as javascript snippet	No	No	High Stateful Ops + Windows to be developed	Medium – only 2 components: Bluemix runtime + MessageHub service, but lots of custom code	Yes	Yes Well established architectural pattern.	Low
OpenWhisk shield dispatcher on Kafka	Yes, shield expressed as javascript snippet	No	No	High Stateful Ops + Windows to be developed	Medium – 4 components: BMX runtime + MessageHub + OpenWhisk + Redis + lots of custom code	Yes	Not evaluated as solution doesn't meet MUST HAVE requirements	Medium
Apache Flink with Kafka connector	Limited: Flink Kafka consumer job/group per shield (consuming the same topic)	No	No	Low	High – Flink is no BMX service, to be deployed as container/VM.	Only on Hadoop on docker	Yes	
Spark Streaming on Kafka	No, all "shields" end up in a big java program (of Stream SQL expression)	No	No	Low	High – Spark service only avail in Dallas, so similar to Flink	Only on Hadoop on docker	Yes	
Agent framework shield dispatcher on MQTT	Shield Java code must be run in Agent Container with explicit "state saving"	No	No	High Stateful Ops + Windows to be developed	High – to be deployed by IoT4I DevOps team	Not known	Yes	
IBM streams	No	No	Yes	Low	Low	Yes	Yes	
ODM Decision Server Insights	Not known	Not known	Yes	Low	Low	No	Yes	
Real Time Insights	Yes	Yes	Yes	Low	Low	Yes	Yes	

Work in progress

THANK YOU!